1-5

SINC-LINK

TIMEX-SINCLAIR USERS CLUB

NEWSLETTER

Toronto, Ontario

PRINTED IN CANADA

B6146

MESSAGE FROM THE PRESIDENT

GREETINGS MEMBERS;

Welcome to our second nesletter of 1984. Our club is growing because of your enthusiasm and word-of-mouth. Keep up the good work! Membership is well over 100 now and growing.

Keep on computing
Greg Lloyd
(President)

CLUB INFORMATION

REMEMBER: Club meetings are held on the FIRST & THIRD Wednesday of each month. NOT every 2nd Wednesday!!!

The club newsletter is published bi-monthly. Therefore, the next publication will be the first meeting in May.

AGENDA FOR UPCOMING MEETINGS

MARCH 7 — How to use the "Hunter Board" with non-volatile memory including a demonstration using the 8K "Hunter Board". The demonstration will include both Basic and Machine Code.

MARCH 21 — How to use the "Hunter Board" as an Eprom Board: includes a demonstration using a fully populated Eprom board with 8K of Eproms. (either four 2K Eproms or two 4K Eproms).

APRIL 4 — An evening in basic. Bring your ZX or TS manuals or any other Basic reference guide in order to refer to during the course of the evening. ALSO, the evening includes an introduction to the different types of keyboards available for the ZX/TS including a course from our own resident keyboard manufacturer.

APRIL 18 — How to use the different "FAST LOAD/SAVE" software and hardware available to us. INCLUDES A DEMO OF QSAVE and ZXLR8.

EXECUTIVE OFFICERS

****

PRESIDENT: *Lloyd Greg*
SECRETARY: *George Chambers*
NEWS EDITOR: *Stan Piotrowski*
MEETING CHAIRMAN: *Harold Goodwin*

TREASURER: *John Roach*
LIBRARIAN: *Martin Mauk*
ACTIVITY DIRECTORS: *Brian Hammond*, *Ian Roberts*
LIASON OFFICER: *Chris Hart*
(out of town members)

ZX-81 COMPUTER JOYSTICK

(by George Chambers)

There have been a number of articles published on fitting a joystick to the ZX-81. Most of these are limited by the fact that they are restricted to a fixed key assignment (usually keys 5 to 8), or work in association with supplementary software. This means that games which have other key assignments require program modificaion. Since the better games are in machine code this is usually out of the question.

I would like to describe a hardware approach I have used to the problem, which provides for complete flexibility of key/joystick assignment. Essentially, it amounts to bringing the 13 keyboard leads from the computer and 10 leads from a joystick out to the terminals of a card edge socket. When the joystick is to be used, the joystick leads are bridged to the appropriate computer leads by means of a small patch card inserted in the card edge socket.

A series of patch cards are made up with cross-connections appropriate to each of the joystick applications (read "games"). To provide the necessary flexibility, i.e. removal of the joystick when not required, it will be necessary to provide a second smaller socket for the joystick connection.

For these two sockets I used what was at hand, a 12-pin card edge socket (10-pin sockets seem uncommon) for the joystick and a 24-pin card edge socket for the patch card assembly. I obtained a Burndy PCS2(?) T6B 24-pin socket suitable for the purpose from Active Surplus, 347 Queen St. W., Toronto.

The question arises as to where to mount these two sockets. In my case, I had equipped my computer with an auxiliary keyboard so I simply mounted both sockets on the keyboard mounting panel. Where this is not possible, consider securing the computer to a base of some sort and fastening the sockets to that. It does not seem practical to secure them to the computer itself; yet it is necessary that they be firmly associated with it.

Critical to the success of the project is the selection of a suitable joystick. I purchased a heavy duty arcade style stick for $20.00 from Ace Computer Supplies Inc., 329 Queen St. W., Toronto. It comes unmounted. The important thing to watch for is that the joystick functions by microswitch operation and not by variable potentiometers. It also makes life easier if the switches are separately wired rather than having a common ground return.

We want both the 10 leads from the joystick and the 13 leads from the computer keyboard to appear on the pins of the patch card socket. Therefore we should cross-connect the joystick socket to a block of 10 pins on the patch card socket. Wire these leads so their numerical appearance at the patch socket corresponds to the lead numbering shown on the joystick schematic. As an example, contacts of the "UP" direction microswitch should appear at pins 1 and 2 of the patching socket; the "DOWN" direction to pins 3 and 4, etc.

If you have an auxiliary keyboard, it will probably be simplest to connect the 13 leads directly to the keyboard. If not, then make a soldered connection to the computer where the sockets for the keyboard ribbon cable appear.

... 2

Joystick Assembly

The joystick came unmounted. I made up a wooden case with outer dimensions of 4 x 6 x 3 inches. The joystick has mounting brackets which facilitate mounting in the case.

Also required for the joystick assembly is a "FIRE" button. I used a door bell type button since they were at hand. The "FIRE" button needs to be mounted on the top of the case convenient for operation. I obtained 3 feet of 10 conductor ribbon cable for the joystick (any cable will do). Watch that it is made up of stranded wire and is reasonably flexible. Cut a slotted hole in the case for the ribbon cable and wire it to the joystick and the "FIRE" button. The other end of the cable needs to be terminated to a 10-pin plug. There is nothing special here. You simply want to have a jack and socket arrangement that can handle 10 conductors. Again, I used what was at hand; I etched a piece of circuit board and fashioned it to plug into the joystick socket. The joystick cable was soldered to the PC board and secured with thread and Epoxy glue. A piece of "vero-board" will also do, except make sure its mating socket has the correct pin spacing (there are several "standard" pin spacings).

Making up the Patch Cards

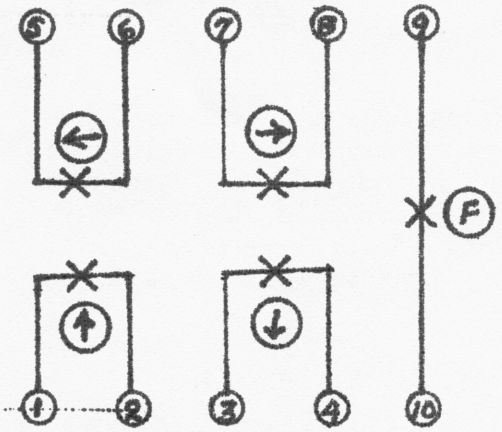
If you elect to use "vero-board" for the patch cards, it is a breeze to make sure your connector has the correct pin spacing to permit this. Otherwise you will need to purchase a scrap of printed circuit board; then cut it in order to plug it into the socket. Leave about 1-1½ inches exposed to allow cross-connections to be soldered and to provide a fingerhold for insertion and removal. I will not get into etching the PC board except to say that the Scotch brand 810 Magic Transparent tape will do a good job of masking the board and etching. Fluid (Etching Acid) can be obtained from any Radio Shack store.

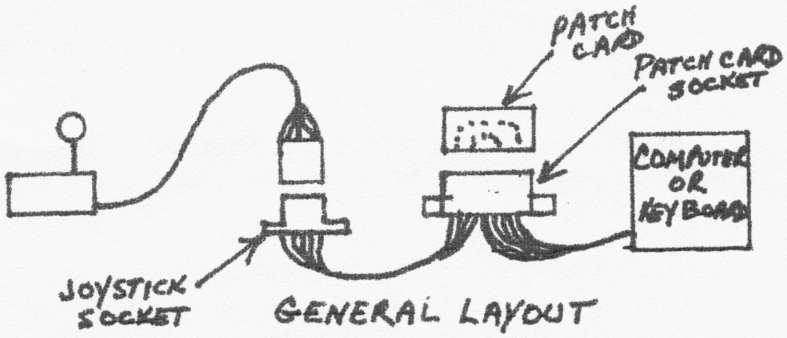
The patch card needs to have cross-connections wired on it to associate the joystick terminals to the appropriate computer leads. I found this assigning process difficult to explain. So difficult, in fact, that I wrote a program to do it for you. To use the program, simply enter the key called for in turn. The program will then display a complete cross-connection list appropriate to that application.

A copy of the program will be available in the club tape library if you are not up to typing this list.

==========

JOYSTICK SCHEMATIC





GENERAL LAYOUT

```
  0 REM   WRITTEN BY GF CHAMBERS
  3 REM
  4 REM    A PROGRAM TO PROVIDE
  5 REM    LEAD ASSIGNMENTS FOR
  6 REM   ZX81 JOYSTICK CONTROL
 60 LET C=0
 70 DIM E$(5,2)
 80 DIM F$(5,2)
 90 DIM E(10)
100 DIM F(10)
110 DIM A(10)
120 DIM G(10)
130 LET D$="  *UP*   *DOUN* *LEFT
*  *RIGHT**FIRE*  "
140 LET A$="12345QUERT09876ASDF
GPOIUY ZXCV LKJH .MNB"
150 CLS
160 PRINT "    ZX81/TIMEX 1000 J
OYSTICK"
170 PRINT "    ================
========"
180 PRINT "         LEAD ASSIGMNE
NTS"
190 PRINT AT 10,0;"PRESS KEY TO
BE  USED  FOR  THE  ,,     ";D$(
C*7+1 TO C*7+7);" DIRECTION"
200 PRINT "    ========="
210 PRINT AT 18,0;"IF THE ";D$(
C*7+1 TO C*7+7);" DIRECTON  IS
NOT  ,,  REQUIRED, PRESS THE ""N/
L"" KEY"
220 LET C=C+1
230 LET I=CODE INKEY$
240 IF INKEY$<>"" THEN GOTO 240
250 IF I=118 AND C=5 THEN GOTO
290
260 IF I=118 THEN GOTO 150
270 IF I<27 OR I>63 THEN GOTO 2
30
280 LET G(C)=I
290 CLS
300 PRINT AT 10,10;"ONE MOMENT"
310 GOSUB 480
330 IF C<5 THEN GOTO 150
340 PRINT "       JOYSTICK X-CONN
ECTIONS"
350 PRINT ,,"  GAME          J/S
    KEYBOARD"
360 PRINT "FUNCTION KEY   LEAD
TO  TERMINAL"
370 PRINT "============  =====  ==
====  ========"
380 LET D=1
390 FOR C=1 TO 10
400 IF INT (C/2)<>(C/2) AND G(D
)=0 THEN PRINT D$((D*7)-6 TO D*7
);"       NO LEADS REQUIRED
410 IF INT (C/2)<>(C/2) AND G(D
)=0 THEN  LET C=C+1
420 IF INT (C/2)<>(C/2) AND G(D
)=0 THEN  GOTO 450
430 IF INT (C/2)<>(C/2) AND G(D
)<>0 THEN PRINT D$((D*7)-6 TO D*
7);"    ";CHR$ G(D);"          ";C;" --
------ ";E$(D, TO 2)
440 IF INT (C/2)=(C/2) AND G(D)
<>0 THEN PRINT TAB 15;C;" ------
-- ";F$(D, TO 2)
450 IF INT (C/2)=(C/2) THEN LET
 D=D+1
460 NEXT C
470 GOTO 9999
480 FOR J=1 TO 40
490 IF I=118 THEN LET A(C)=0
500 IF I=CODE A$(J) THEN LET A(
C)=J
510 IF I=CODE A$(J) THEN LET J=
40
520 NEXT J
530 LET E(C)=INT ((A(C)+4)/5)
540 LET F(C)=A(C)+4-(E(C)*5)
550 LET E$(C, TO 2)="D"+STR$ E(
C)
560 LET F$(C, TO 2)="K"+STR$ F(
C)
570 CLS
580 RETURN
9800 RAND USR 14336
9810 SLOW
9820 CLS
9830 RUN
9900 SAVE "JOYSTIC"
9910 RUN
9999 STOP
```

SAMPLE PRINTOUT FOR MAZOGS

JOYSTICK X-CONNECTIONS

GAME FUNCTION	KEY	J/S LEAD	TO	KEYBOARD TERMINAL
UP	W	1	---------	D2
		2	---------	K1
DOWN	S	3	---------	D4
		4	---------	K1
LEFT	A	5	---------	D4
		6	---------	K0
RIGHT	D	7	---------	D4
		8	---------	K2
FIRE	U	9	---------	D6
		10	---------	K4

In the last Newsletter we started a D-Base type program; we set up the Menu, Dimensioned all the variables, set up the capabilities to add data and list all the data.

Let's continue with the next important capability. We can add data and when that data is out of date or no longer required, we need to be able to delete it from our Data Base. If you remember from our last discussion, all the branching menu choices occur at lines of increments of 1000. For example, choice #1 is located at 1000, choice #2 at 2000, etc. The Delete choice, therefore, will begin at line #2000. Add the following lines to your program and they will be explained at the end.

```
2000 FAST
2010 CLS
2020 PRINT AT 6,0;"ENTER RECORD
NUMBER TO DELETE:"
2030 SLOW
2040 INPUT X$
2050 FAST
2060 IF X$<"1" OR X$>STR$ N THEN
 GOTO 2200
2065 CLS
2066 LET I=VAL X$
2070 GOSUB 20
2080 LET I=VAL X$
2090 FOR I=Z TO N-1
2100 LET R$(I)=R$(I+1)
2110 NEXT I
2120 LET R$(N)=""
2130 LET N=N-1
2140 PRINT AT 3,0;"RECORD ";X$;"
 HAS BEEN DELETED"
2150 GOTO 2230
2200 FAST
2210 CLS
2220 PRINT AT 6,2;"RECORD ";X$;"
 IS NOT ON FILE"
2230 PRINT AT 20,2;"DELETE ANOTH
ER RECORD (Y/N)"
2240 PAUSE 4E4
2250 IF INKEY$="Y" THEN GOTO 200
0
2250 GOTO 100
```

Lines 2000 to 2030 are self-explanatory. Notice in line 2040 we used x$ instead of an X to wait for a number. This is another kind of "Idiot Proofing"; if we used an X which must have a number entered (no characters whatsoever) then any other character other than a number would have caused the program to stop with an error code. For people, or even for you, if the error code is not handled properly, you would have lost all your data. If this ever happens, always by direct command "GOTO 100" and no data is lost. (That's why it is always wise to have 1 type of "skeleton" setup so that you can always get back to the Menu without having to remember where the Menu is in each program).

To continue, 2060 is another "Idiot Proof" statement. If your number entered is less than 1 or greater than the records currently on file then the routine jumps to line 2200 to inform you of this. Without this line, the computer would attempt to Delete a file that does not exist. Note also in line 2050; "STR$". By now, you must realize that every time you see the $ sign, you know we are dealing with strings. Strings can be characters or numbers whereas a letter such as "X" can only represent a numerical value (remember your Algebra days?).

Let's discuss STR$ and the opposite computer function of VAL. First of all, X$ is used in line 2060 so in order for it to compare whether the following string is larger or smaller, it must be followed by a string. The record number is actually converted for this line only, to a string value. The opposite happens with VAL with one exception; the characters following VAL must be numerical only and will be contained in quotes. Therefore, to sum up, the STR$ function asks the argument following it to be a numerical value to be converted to a string; the VAL will be followed by a string to be converted to a numerical value, e.g. PRINT VAL "1000". You see this a lot in 1K programs because each numerical value takes up 5 bytes of memory plus a byte for each digit while a string takes up only one byte for each character whether it is a number or a character. Line 2066 converts the value of your Record number which is X$ to a real or a number value, in this case "Z".

The next little routine is the actual deletion so let's go through it slowly. (Lines 2066 and 2070 are what's called in computer programming technology ... a stupid mistake. They don't do anything except confuse the novice programmer which I apologize for). The next line (2090) is a FOR/NEXT loop that starts with the Record you want deleted to the Record number currently on file minus 1. For example, if we have 10 records on file and we want to delete Record 5, then the loop starts off with 5 to N (record number)-1 or, in this case, to Record 9. Since we are deleting 1 record, there will only be 9 records instead of 10. Line 2100 does the exchange. To continue with the example, 2100 says R$(5) will now equal R$(5+1) or R$(5) will equal R$(6). Follow the logic carefully - whatever is on the left side of the "=" sign will not be whatever is on the right side of the "=" sign. If R$(5) or Record #5 was Mr. Jones and R$(6) was Mr. Smith, R$(5) would now be Mr. Smith and the record with Mr. Jones would no longer exist. If the computer was stopped right now, record #6 would also be Mr. Smith. But the loop increases by 1 and R$(6) would contain the string or record of R$(7) and so on up to the end of our example of 9 where Record #9 would now contain whatever was in R$(10). Remember, N-1- Again, remember that R$(9) has now become whatever was in R$(10) so that R$(9) and R$(10) have the same data; hence line 2120 which declares R$(N) which is the current number of records on file or, in our example, the 10th record becomes an empty string. Two quotations following each other with no spaces or characters means an empty string. Line 2130 now makes sure that since we have deleted a file, we can make use of the empty file so it subtracts 1 from the current record number; in our example again, we had 10 records, deleted 1 record so 1 is substracted. Currently N=10 and N-1 means N=9. Line 2140 advises the user, his selected record number has been deleted then goes on to ask if you want any more records deleted.

If you wish to "Idiot Proof" a bit more, you can use the previous mentioned "stupid mistake" lines. By adding one more line such as: 2075 PRINT "IS THIS THE RECORD YOU WANT DELETED (Y/N)?" and the lines that must follow follows through with the deletion if "Y" or goes back to the Menu if "N". This is in case you entered the wrong number by mistake. You should be able to add those lines yourself by now so give it a shot; the computer won't blow up if you're wrong. The remaining lines for the deletion section are self explanatory.

In order to SAVE the program with data, we enter choice #6 which goes to line 6000. Add the following lines to your program; (start your tape to SAVE then press "6").

```
6000 FAST
6010 CLS
6020 SAVE "NAME FILE"
6030 GOTO 100
```

Just a short explanation of these lines. First of all, any Basic program you list, do you ever wonder why the last character of the SAVE name is inverted? This is used by the machine code which helps in its speed and recognition. It reads each character until it sees the inverse character and "knows" that it is the end of the SAVE name. Upon LOADing at a different time, it jumps to the Menu keeping all the data and variables intact. If 6030 has said RUN, then this command resets all variables to nothing (does not exist) and everything must be redimensioned and there obviously would be no data on file. (CLEAR does the same thing). This is why suggestions are made to "CLEAR" the variables (if they are of no value) prior to SAVEing a program. In our program, if all variables are CLEARed, the program would SAVE in 2 to 3 minutes while otherwise it could take up to 7 minutes to SAVE and LOAD. Of course if it was CLEARed prior to SAVEing, we would lose all our data which we don't want.

Well that's is for this Newsletter in programming in Basic. If you need further explaining, let me know or if you need answers or answers to specific questions about Basic programming, again let me know or any other of our more experienced members which include most of the executive officers.

==========

MACHINE CODE PROGRAMMING

Beginning with this Newsletter, we will learn Machine Code programming from scratch and continue in the following Newsletters ending with a small "Shoot-Em-Up" type Space Invaders game with full explanations so that you will be able to make your own Machine Code programs such as games.

First of all we ask the question, Why Machine Code? If by now you don't know, Machine Code can increase the speed, the execution, literally up to about 1000 faster. If you have ever programmed an "X" or any other character to move across the screen in Basic, you can see how agonizingly slow it is. In Machine Code, you have to actually slow the movement down by as much as 10,000 times ... notice ten thousand because machine code is so fast.

This, and the following articles, assume you have some knowledge of Basic since we will be doing some comparisons to help you in understanding what is going on.

The lowest form of a computer language is the Binary system. We really don't have access to this form but the next level which is using the decimal or hexadecimal system but we stillhave to understand the Binary system in order to understand smoe machine code processes. In the decimal system is it composed of

10's and increment of tens. For example, if we begin counting, we start with 0, 1, 2, etc. and when we reach 9, the ones column revert to zero and the tens column contains a one. (10). The same again happens when we reach the number 99. The ones and tens columns revert to zeroes and the hundreds column now contains a one. The binary system is much different. It has only two states or can only hold 2 numbers in each column instead of ten. That is because the microprocessor (the Z80 in the ZX-81), the Timex 1000 or any of the other microprocessors in all computers) has only 2 states - on or off or high/low voltages. As you can see the simplest form of the computer down to the bare essentials is on/off. This means that every household actually contains a very primitive form of a computer (although it may seem ridiculous) is your switch to turn on the kitchen lights, for example (on/off). But a computer has millions of these tiny switches (electronic signals in actual fact) in a chip ½ x 2 inches. Therefore, to count we begin with 0 (as in the decimal system), then 1 and that's all the ones column can count to so as in the decimal system, the next column which was a zero now becomes a one and the ones column reverts to zero. We have thus counted to 2 which looks like this in binary: 0000 0010. To count to three, the binary number becomes: 0000 0011 which now has a 1 in the ones column. To continue to 4, the ones columns have counted as high as they can in binary so the 3rd row is increased by 1 and the first 2 rows revert back to zero (again as the same condition exists for the decimal system). The number 4 in binary then looks like this: 0000 0100 and so on. If you notice, we have 8 rows or columns. Each column is called a "BIT" and 4 bits make a "NIBBLE". 8 bits or 2 nibbles makes one byte (which comes from Binary Digit). The Z80 is such that 8 bits can travel in parallel to and from the microprocessor so our system is using an "8-bit microprocessor".

Some home computers have "16-bit" microprocessors. What does this mean? Speed!!! The microprocessor or CPU can receive and send data 16 bits at a time. To explain this, imagine an 8 lane highway and we have gravel trucks containing gravel (computer data). If we send them down the highway in single file, there would be a considerable time interval between the first truck arriving at its destination and the last truck. Imagine, now, if we had a 16 lane highway; there would be a considerable increase in time differences or speed in which the data arrived. Now you can see why we have ribbon cables coming from computers. Besides the signal to turn a device such as a disk drive, on and off, it also is capable of sending and receiving data 8 bits at a time (or more in a 16-bit system) - remember the 8 lane highway). If you "turn on" all the bits (make them all ones) you will see they add up to 255 (check the last Newsletter for the Binary-Hex-Decimal printout). Therefore, each byte can only hold any decimal value from 0 to 255 (or 256 numbers - remember zero is a number too).

Always remember when looking at a binary number, we read it the same as a decimal number - from right to left, e.g. 239 has 9 ones, 3 tens and 2 hundreds.

0000 1010 has no ones, 1 two, no fours and one 8 so this number is 2+8 or 10 decimal.

There, that wasn't too complicated. Understanding this will actually make the rest easier to understand.

We use the decimal system or hexadecimal system to represent these binary numbers since they can become unwieldy. Can you imagine trying to read:

0100 1010
1101 0001
etc.

We must be able to use, manipulate and change these values in a systematic or logical order and able to locate them or place them anywhere we want (within reason) so we have what are called "addresses". Each address (beginning with zero) is capable of holding 1 byte (or any one decimal number from 0 to 255). 1024 of these bytes together make up 1K of memory (in actual fact, 1024 addresses means 1K). Therefore, 16K has actually 16383 locations or addresses. If you multiply 16 x 1024 your answer will be 16384 but don't forget, zero is the first address so from 0 to 16383 are 16,384 addresses.

Each and every computer must be able to communicate with the outside world such as checking if a key is being pressed or the letter "A" is printed on the TV screen. This operating system is called the "ROM" or Read Only Memory. We can use it but cannot change the contents so the letter "A" will always look the same, etc. In the ZX-81 the ROM uses for itself, addresses 0 to 8191. There are many types of ROM's such as COBOL or FORTRAN but the ZX-81 has the language BASIC built into it. What Basic is, or any other computer language, is an elaborate machine code process. That is the reason that Basic is so slow as compared to machine code. First it takes the data, checks it for Syntax (Rules of the ROM) then checks to see what each of the commands means in machine code, translates it to machine code, executes the command, translates it back to Basic and displays the results (if required). Again, let's use another analogy. Suppose you bought a Stereo Component set in kit form BUT all the instructions are in Chinese. You would translate each instruction, act on it, then continue on to the next instruction. But what if it referred you back to a previous instruction? You would have to translate it again and act on it. This is like Basic. Time consuming but it works.

A much better way would be to sit down and translate the whole thing then build it. This is the same as Compiler programs that convert Basic program to Machine Code. A much, much faster way. But what if you had built 100,000 sets? You could build the next one without even looking at the instructions. This is machine code. No referring to any routines or instructions but just execute each command in turn or each step, as it came along.

You can see that the ZX-81 is actually a 24K computer (if we follow the advertisements as with other computers). We have an 8K ROM and 16K RAM pack which adds up to 24K. To find out why the 8-Bit computers can only access up to 64K we return again to the binary numbers. This point is now important since we will be using this concept in a lot of our machine code routines.

As pointed out earlier, 1 byte can only hold a decimal number of 255 so the microchip inventors made it such that under certain conditions (to be discussed at length later), it can use a combination of 2 bytes following each other. For example, under a certain condition the computer is told that a number larger than 255 is required so it takes the first byte it receives then waits for the next byte. The next byte it receives is multiplied by 256 then added to the

first byte (remember 0-255 are 256 numbers). For example, if the first byte the computer receives is 10 and the second byte it receives is 5, the resulting number is:

$$10 + (5 * 256) = 1290$$

Now taking this step to the end of possibilities: the first number is 255 and the second number is 255;

$$255 + (255 * 256) = 65535$$

Therefore from 0 to 65535 are 65536 addresses. Since 1K is 1024, we divide: 65536/1024 is 64 so an 8-bit CPU can access up to 64K of memory. This sounds like a lot of memory for you to use but the user does not really have access to all the memory. In one particular computer, the ROM takes up 12K, the Disk Operating System (DOS) takes up a further 10K, hi-res graphics, printers and other devices take up more memory leaving the user with 35K of memory for Basic or machine code.

In the ZX-81, we are even less fortunate with having a 64K RAM pack if you are thinking that you have more memory for your Basic program. The ZX-81 is set up that we can still only use 16K for Basic but part of the 64K RAM pack is used for the systems variables and your variables so that you do have more memory to use, the remainder of the 64K can only be accessed by using machine code. Likewise if you have the Hunter board that occupies the memory locations between 8192 and 16383. When you buy the ZX-81 with a 16K RAM pack the ROM occupies memory locations 0 to 8191 and the RAM pack accesses memory locations 16384 to 32768 which leaves the area blank used by the Hunter board (which is an extra 8K) but can only be used by machine code.

Further checking your manual, you will find that the ZX-81 uses some of your RAM for itself which it calls the "System Variables" (from address 16384 to 16513). The computer has to know, for example, how many lines are presently being used on the screen so that it knows when the TV screen is filled to the end with data or how much memory you have available for Basic. Some of these variables can be changed to suit your needs without causing the ZX-81 to "crash" but others cannot be manipulated. For example, if you check the system variables, you will find that addresses 16388 and 16389 hold the location of RAMTOP. Now that you understand how these 2 addresses can hold a number beyond 255, you can PEEK these locations. You will find (for 16K) that 16388 is 0 (PRINT PEEK 16388) and 16389 is 128. Therefore, 0 + (128 * 256) = 32768. If you go back and check our previous discussion, you will find that the 16K RAM pack uses addresses 16384 to 32768 ... there it is, the end of RAM or RAMTOP. Now, by POKEing various numbers into these 2 addresses we can fool the computer into thinking it has less memory (lowering RAMTOP), we can make use of the area above RAMTOP for machine code routines and NEW or CLEAR does not alter the code above RAMTOP because it checks 16388 and 16389 to see how much memory is available. It then clears only this area. After the above lengthy discussion, you should be able to check and see how much memory the programmer wants to use by POKEing various numbers to "Lower RAMPTOP".

Now we get to the address that is the first address the programmer can use, whether for machine code or Basic and that is 16514. If you recall your Basic

you know that by placing a REM statement anywhere in your program is a REMark statement that the computer does not act on or do anything with it. It just tells the Basic programmer what is going on. For example, 10 REM THIS IS MY PROGRAM WRITTEN BY, etc. Carrying this concept one step further, that line does occupy memory addresses, one for each character. If you add the following program lines you will find it as shown;

```
10 REM  THIS IS MY PROGRAM
20 FOR I=16514 TO 16531
30 PRINT I;","; PEEK I, CHR$ PEEK I
40 NEXT I
50 STOP
```

After runnning this program, you will find that the REM statement uses 18 address, or memory locations or bytes. So if we continue another step, we can put any decimal number in those addresses after the REM and it won't make any difference to the Basic program. So we can, therefore, use these addresses for our machine code routines. Without a REM statement, we cannot arbitrarily POKE numbers into the addresses without causing a "crash" so by placing a REM statement at the start of the Basic area, namely 16514, we can place any number of characters after it to save the addresses to be used for our machine code routines. We can enter any character we want such as: 123456 or a number of X's so long as there are as many as or more than the number of bytes in the machine code routine.

Using a REM gives you the capabilities to SAVE the routine without any fuss whereas above RAMTOP it is much more difficult. But what if you needed a hundred or a couple of thousand bytes reserved; that's a lot of extra typing. Fortunately, we follow that computer axiom, "LET THE COMPUTER DO THE WORK". The following is a Machine Code routine that prints the line number, the word "REM" and the number of X's chosen by the user. Notice that most machine code routines are the first line in any Basic program. That is so that we don't have to calculate the address to be used by the machine code routine since we know that the first address is always 16514.

First we must lower RAMTOP: POKE 16388,186 POKE 16389,127 NEW, NEWLINE. In order to enter any machine code, we must turn back to Basic to "load" the machine code so enter the following lines (called a "loader program"):

```
10  SCROLL
20  PRINT "ENTER STARTING ADDRESS"
30  INPUT X
40  SCROLL
50  SCROLL
60  PRINT "ENTER YOUR MACHINE CODE"
70  SCROLL
80  SCROLL
90  INPUT Y
100 POKE X,Y
110 PRINT X;"'"; PEEK X
120 LET X=X+1
130 GOTO 80
```

REM Generator Machine Code:

RUN the program and enter the address: 32699

Enter the following bytes (not the commas):

237, 75, 66, 64, 205, 231, 2, 197, 3, 3, 197, 1, 6, 0, 42, 41, 64, 9, 34, 41, 64, 33, 125, 64, 205, 158, 9, 35, 112, 35, 112, 35, 193, 113, 35, 112, 35, 54, 234, 35, 54, 118, 193, 197, 205, 158, 9, 193, 197, 35, 54, 61, 11, 120, 177, 32, 248, 193, 42, 41, 64, 9, 34, 41, 64, 195, 7, 2

NEW your "loader" program since it has done its job then enter the following Basic lines:

```
10  PRINT at 10,0; "HOW MANY BYTES FOR THE REM STATEMENT?"
20  INPUT X
30  POKE 32711, X- INT (X/256) * 256
40  POKE 32712, INT (X/256)
50  RAND USR 32699
60  CLS
70  LIST Ø
```

Well, that' i for this Newsletter about machine code. Hopefully you have understood the basic concepts since it will give you a better understanding of the articles to follow.

As I've said before, please let me know what kind of articles you wish to see in your Newsletter. I am trying to give you a little of the various aspects of the ZX-81 to help you in programming a computer, especially the ZX-81.

==========

AN ARTICLE SUBMITTED BY GEORGE CHAMBERS

The University of Wisconsin-Parkside hold annual computer problem solving contests. These contests have several levels of difficulty, namely, Elementary, Junior and Senior. Perhaps you would like to try one. I am including one from each division as used in the 1982 competition.

1. Elementary Division

LETTER HOME

You are away at summer camp and you have run out of money. You plan to write a letter home to ask for more. Everyone else at the camp is in the same situation. Since you are learning how to program a computer, you decide to write a program that will generate a letter that anyone in your condition can use. The contents of the letter it up to you but it must include certain pieces of information that are supplied by the user of the program. This information is underlined in the following sample letter which you may use:

DEAR MOM AND DAD

THE $15.00 YOU GAVE ME FOR SPENDING MONEY AT CAMP IS GONE. I SPENT MOST OF IT ON SNACKS. DO YOU THINK YOU COULD SEND AN EXTRA $5.00.

THINKING OF YOU OFTEN.

LOVE,
KAREN

Write the program that asks for all the underlined information and prints out a letter home. It is not important that your letter looks exactly like this one but it must contain the same information.

2. Junior Division

TRIANGULAR DESIGN

The triangle listed below is generated by following a certain algorithm. Your job is to discover this algorithm and use it to write a program which will generate similar triangles for any number of rows:

```
1
232
34543
4567654
567898765
67890109876
7890123210987
```

Test your program for rows N=7, and N=14.

3. Senior Division

FIFTEEN

Write a program to input a string of 5 digits (Ø - 9) and to find and print all possible combinations of these digits which add to 15. Use the digit Ø to count as 10 in computing the combinations. For example, with the input: 5Ø154, the output should display the following (perhaps in some other order):

```
5 Ø 1 5 4
- - - - -
5 Ø
5   1 5 4
  Ø 1   4
  Ø   5
```

In this example, there are four combinations totalling 15. Test your program with each of the following inputs:

(1) 50154
(2) 78787
(3) 55555
(4) Ø6528

==========

T.T.S.U.C. MEMBERSHIP SURVEY

Equipment owned: Quantity

T/S 1000___ 16K___ 32K___ 64K___ Add on ROM board___ 2040 Printer___
Other Printer___ Modem___ Large Keyboard___ Joystick___ Disk and Controller___

Other _____
Tapes: homegrown_____ from magazines_____ commercial_____

Background:

Beginner 1 (where do all these plugs go?)_____
Beginner 2 (how do you code a FOR-NEXT loop?)_____
Beginner 3 (my program works, but no one can read them)_____
Beginner 4 (what's a LDIR command?)_____
Beginner 5 (why can't this machine have a decent i/o structure like my VAX?)_____

Why are you with us? (more than one answer is OK)

To learn BASIC___ To learn machine code___ To meet others that are working on this machine___ To learn about new equipment___ To learn about new programs___ To learn how to use commercial programs___

Other _____

What kind of club do you want?

The biggest in Canada___
The most helpful in Canada___
The friendliest in Canada___

Other _____

How would you change the meetings?

They're perfect___ More product demo's___ More program demo's___ More info for beginners___ More info for advanced___ Separate small interest groups (sig's)___ More time to talk to those around me___ More tutorials___

Other _____

How would you change the Newsletter?

It's perfect___ More general articles___ Have program reviews___ More book reviews___ More hardware: news___ More tutorials___ News from other clubs___ Advertising___ Program mods___

Other _____

What will you do to achieve the above?

Nothing___ Help on committees if called___ Write for the newsletter___ Volunteer for committees___ Start committees___ Coordinate committees___

Other _____

Potential committees (and existing ones)

BASIC Education___ M/C Education___ Newsletter___ Library___ Inter-Club Communication___ News Advertising___ Publicity___ MEETING PLANNING___ Membership___ Contests___

Skill Level:

Yours 0..1..2..3..4..5..6..7..8..9
Meetings 0..1..2..3..4..5..6..7..8..9
Newsletter 0..1..2..3..4..5..6..7..8..9

Name and address if you would like to help

_____
_____
_____

(PLEASE RETURN TO EXECUTIVE MEMBER OF TIMEX-SINCLAIR USER CLUB)